NCSA httpd tutorials

In an effort to make this documentation a bit more usable, we have written these tutorials on various aspects of server setup.

- ## NCSA httpd directory indexing

  NCSA httpd provides a directory indexing format which is similar to that which will be offered by the WWW Common Library. To set up this indexing, follow these steps.

  For an example of what these indexes look like, take a look at the demo.

  ### Activating Fancy indexing

  You first need to tell httpd to use the advanced indexing instead of the simple version. The simple version should be used if you prefer its simplicity, or if you are serving files off of a remote file server, for which the stat() call would be costly. You tell the server which you want to use with either the IndexOptions directive, or the older FancyIndexing directive. We recommend:

  ```
  IndexOptions FancyIndexing
  ```

  ### Icons

  NCSA httpd comes with a number of icons in the /icons subdirectory which are used for directory indexing. The first thing you should do is make sure your Server Resource Map h as the following line in it:

  ```
  Alias /icons/ /usr/local/etc/httpd/icons/
  ```

  You should replace `/usr/local/etc/httpd/` with whatever you set ServerRoot to be.

  Next, you need to tell the server what icons to provide for different types of files. You do this with the AddIcon and AddIconByType directives. We recommend something like the following setup:

  ```
  AddIconByType (IMG,/icons/image.xbm) image/*
  AddIconByType (SND,/icons/sound.xbm) audio/*
  AddIconByType (TXT,/icons/text.xbm) text/*
  ```

  This covers the three main types of files. If you want to add your own icons, simply create the appropriately sized xbm, place it in /icons, and choose a 3–letter ALT identifier for the type.

  httpd also requires three special icons, one for directories, one which is a blank icon the same size as the other icons, and one which specifies the parent directory of this index. To use the icons in the distribution, use the following lines in srm.conf:

  ```
  AddIcon /icons/menu.xbm ^^DIRECTORY^^
  AddIcon /icons/blank.xbm ^^BLANKICON^^
  AddIcon /icons/back.xbm ..
  ```

However, not all files fit one of these types. To provide a general icon for any unknown files, use the DefaultIcon directive:

```
DefaultIcon /icons/unknown.xbm
```

Descriptions

If you want to add descriptions to your files, use the AddDescription directive. For instance, to add the description "My pictures" to `/usr6/rob/public_html/images`, use the following line:

```
AddDescription "My pictures" /usr6/rob/public_html/images/*
```

If you want to have the titles of your HTML documents displayed for their descriptions, use the IndexOptions directive to activate `ScanHTMLTitles`:

```
IndexOptions FancyIndexing ScanHTMLTitles
```

WARNING: You should only use this option if your server has time to spare!!! This is a *costly* operation!

Ignoring certain items

Generally, you don't want httpd sending references to certain files when it's creating indexes. Such files are emacs autosave and backup files, httpd's .htaccess files, and perhaps any file beginning with . (if you have a gopher or FTP server running in that directory as well). We recommend you ignore the following patterns:

```
IndexIgnore */.??* */README* */HEADER*
```

This tells httpd to ignore any file beginning with ., and any file starting with README or HEADER.

Creating READMEs and HEADERs

When httpd is indexing a directory, it will look for two things and insert them into the index: A header, and a README. Generally, the header contains an HTML <H1> tag with a title for this index, and a brief description of what's in this directory. The README contains things you may want people to read about the items being served.

httpd will look for both plaintext and HTML versions of HEADERs or READMEs. If we add the following lines to srm.conf:

```
ReadmeName README
HeaderName HEADER
```

When httpd is indexing a directory, it will first look for HEADER.html. If it doesn't find that file, it will look for HEADER. If it finds neither, it will generate its own. If it finds one, it will insert it at the beginning of the index. Similarly, the server will look for README.html, then README to insert a trailer for the document.

- ## Setting up CGI in NCSA httpd

CGI scripts are a way for documents to be generated on the fly. You should first read the brief introduction to CGI to learn what it is and why you would want to use it.

There are two main mechanisms to tell NCSA httpd where your scripts are. Each has its pluses and its minuses.

ScriptAlias

The first approach is based on the Server Resource Map directive ScriptAlias. With this directive, you specify to the server that you want to designate a directory (or directories) as script–only, that is, any time the server tries to retrieve a file from these directories it will execute the file instead of reading it.

The usual setup is to have the following line in srm.conf:

```
ScriptAlias /cgi-bin/ cgi-bin/
```

This will make any request to the server which begins with /cgi–bin/ be fulfilled by executing the corresponding program in ServerRoot/cgi–bin/.

You may have more than one ScriptAlias directive in srm.conf to desingnate different directories as CGI.

The advantage of this setup is ease of administration, and centralization. Many system managers don't want things as dangerous as scripts anywhere in the filesystem. The disadvantage is that anyone wishing to create scripts must either have their own entry in srm.conf or must have write access to a ScriptAliased directory.

CGI as files

NCSA httpd 1.2 allows you to create CGI scripts anywhere, by specifying a "magic" MIME type for files which tells the server to execute them instead of sending them. To accomplish this, use the AddType directive in either the Server Resource Map or in a per–directory access control file.

For instance, to make all files ending in .cgi scripts, use the following directive:

```
AddType application/x-httpd-cgi .cgi
```

Alternatively, you could add .sh and .pl after .cgi to allow automatic execution of shell scripts and PERL scripts. Note that you have to have Options ExecCGI activated in the directory you create scripts. (you might want to read more about directives like Option in the docs for installation).

The advantage of this setup is that scripts may be absolutely anywhere. The disadvantage is that scripts may be absolutely anywhere (especially places you don't want them to be like users' home directories).

- ## NCSA httpd server side includes

NCSA httpd allows users to create documents which provide simple information to clients on the fly. Such information can include the current date, the file's last modification date, and the size or last modification of other files. In its more advanced usage, it can provide a powerful interface to CGI and /bin/sh programs.

Issues

Having the server parse documents is a double edged sword. It can be costly for heavily loaded servers to perform parsing of files while sending them. Further, it can be considered a security risk to have average users executing commands as the server's User. If you disable the exec option, this danger is mitigated, but the performance issue remains. You should consider these items carefully before activating server–side includes on your server.

Setting Up Includes

First, you should decide which directories you want to allow Includes in. Most likely this will not include users' home directories or directories you do not trust. You should then decide, of the directories you are allowing includes in, which directories are safe enough to use exec in.

For the directories in which you want to fully enable includes, you need to use the Options directive to turn on the option `Includes`. Similarly for the directories you want crippled (no exec) includes, you should use the option `IncludesNOEXEC`. In any directory you want to disable includes, use the Options directive without either option.

Next, you need to tell the server what filename extension you are using for the parsed files. These files, while very similar to HTML, are *not* HTML and are thus not treated the same. Internally, the server uses the magic MIME type `text/x-server-parsed-html` to identify parsed documents. It will then perform a format conversion to change these files into HTML for the client. To tell the server which extension you want to use for parsed files, use the AddType directive. For instance:

```
AddType text/x-server-parsed-html .shtml
```

This makes any file ending with .shtml a parsed file. Alternatively, if you don't care about the performance hit of having all .html files parsed, you could use:

```
AddType text/x-server-parsed-html .html
```

This would make the server parse all .html files.

Converting your old INC SRV documents to the new format

You should use the program `inc2shtml` in the `support` subdirectory of the httpd distribution to translate your documents from httpd 1.1 and earlier to the new format. Usage is simple: `inc2shtml file.html > file.shtml`.

The Include format

All directives to the server are formatted as SGML comments within the document. This is in case the document should ever find itself in the client's hands unparsed. Each directive has the following format:

```
<!--#command tag1="value1" tag2="value2" -->
```

Each command takes different arguments, most only accept one tag at a time. Here is a breakdown of the commands and their associated tags:

- `config`

  The config directive controls various aspects of the file parsing. There are two valid tags:

  - `errmsg` controls what message is sent back to the client if an error includes while parsing the document. When an error occurs, it is logged in the server's error log.

  - `timefmt` gives the server a new format to use when providing dates. This is a string compatible with the `strftime` library call under most versions of UNIX.

  - `sizefmt` determines the formatting to be used when displaying the size of a file. Valid choices are `bytes`, for a formatted byte count (formatted as 1,234,567), or `abbrev` for an abbreviated version displaying the number of kilobytes or megabytes the file occupies.

- `include`

  include will insert the text of a document into the parsed document. Any included file is subject to the usual access control. This command accepts two tags:

  - `virtual` gives a virtual path to a document on the server. You must access a normal file this way, you cannot access a CGI script in this fashion. You can, however, access another parsed document.

  - `file` gives a pathname relative to the current directory. ../ cannot be used in this pathname, nor can absolute paths be used. As above, you can send other parsed documents, but you cannot send CGI scripts.

- `echo` prints the value of one of the include variables (defined below). Any dates are printed subject to the currently configured `timefmt`. The only valid tag to this command is `var`, whose value is the name of the variable you wish to echo.

- `fsize` prints the size of the specified file. Valid tags are the same as with the `include` command. The resulting format of this command is subject to the `sizefmt` parameter to the `config` command.

- `flastmod` prints the last modification date of the specified file, subject to the formatting preference given by the `timefmt` parameter to `config`. Valid tags are the same as with the `include` command.

- `exec` executes a given shell command or CGI script. It must be activated to be used. Valid tags are:
    - `cmd` will execute the given string using /bin/sh. All of the variables defined below are defined, and can be used in the command.
    - `cgi` will execute the given virtual path to a CGI script and include its output. The server does not perform error checking to make sure your script didn't output horrible things like a GIF, so be careful. It will, however, interpret any URL Location: header and translate it into an HTML anchor.

Variables defined for Parsed Documents

A number of variables are made available to parsed documents. In addition to the CGI variable set, the following variables are made available:

- `DOCUMENT_NAME`: The current filename.
- `DOCUMENT_URI`: The virtual path to this document (such as /~robm/foo.shtml).
- `QUERY_STRING_UNESCAPED`: The unescaped version of any search query the client sent, with all shell–special characters escaped with \.
- `DATE_LOCAL`: The current date, local time zone. Subject to the `timefmt` parameter to the `config` command.
- `DATE_GMT`: Same as DATE_LOCAL but in Greenwich mean time.
- `LAST_MODIFIED`: The last modification date of the current document. Subject to `timefmt` like the others.

Making your setup more secure

When configuring the access control for your server, you will want to make sure you do not give any unauthorized access to anyone. Please follow these guidelines to ensure that your server is not compromised.

- A word of caution on DNS based access control and user authentication

  The access control by hostname and Basic user authentication facilities provided by httpd are relatively safe, but *not bulletproof*. The user authentication sends passwords across the network in plaintext, making them easily readable. The DNS based access control is only as safe as DNS, so you should keep that in mind when using it. Bottom line: If it absolutely positively cannot be seen by outside people, you probably should not use httpd to protect it.

- Disable Server–side includes wherever possible

  Whenever you can, use the Options directive to disable server–side includes. At the very least, you should disable the exec feature. Note that because the default value of Options is All, you should include an Options directive in every Directory cl ause in your global ACF and in every .htaccess file you write.

- Use AllowOverride None whe rever possible

  Use this directive to prevent any "untrusted" directories (such as users' home directories) from overriding your settings (and thus allowing their friends to execute xterms as nobody with a server−side include or other such horrors). You also gain a bonus in performance.

- Protect your users' home directories

  Protect your users' home directories with `Directory` directives. If your users all have their home directories in one physical location (such as `/home`), then this is easy:

  ```
  <Directory /home>
  AllowOverride None
  Options Indexes
  </Directory>
  ```

  If they are not all in one location such as /home, then you should use this wildcard pattern to secure them (assuming your UserDir is se t to `public_html`):

  ```
  <Directory /*/public_html*>
  AllowOverride None
  Options Indexes
  </Directory>
  ```

  In addition, if you wish to give your users the ability to create symbolic links to things only they own, use the Option `SymLinksIfOwnerMatch`.

- ## Mosaic User Authentication Tutorial

  ### Introduction

  This tutorial surveys the current methods in NCSA Mosaic for X version 2.0 and NCSA httpd for restricting access to documents. The tutorial also walks through setup and use of these methods.

  Mosaic 2.0 and NCSA httpd allow access restriction based on several criteria:

  - Username/password−level access authorization.
  - Rejection or acceptance of connections based on Internet address of client.
  - A combination of the above two methods.

  This tutorial is based heavily on work done by Ari Luotonen at CERN and Rob McCool at NCSA. In particular, Ari wrote the client−side code currently in Mosaic 2.0, and Rob wrote NCSA httpd.

Getting Started

Before you can explore access authorization, you need to install NCSA httpd 1.0a5 or later on a Unix machine under your control, or get write access to one or more directories in a filespace already being served by NCSA httpd.

You also need to be running Mosaic for X version 2.0 or later, or another browser known to support HTTP/1.0–based authentication.

Prepared Examples

Following are several examples of the range of access authorization capabilities available through Mosaic and NCSA httpd. The examples are served from a system at NCSA.

Simple protection by password.

> This document is accessible only to user `fido` with password `bones`.
>
> **Important Note**: There is *no* correspondence between usernames and passwords on specific Unix systems (e.g. in an `/etc/passwd` file) and usernames and passwords in the authentication schemes we're discussing for use in the Web. As illustrated in the examples, Web–based authentication uses similar but *wholly distinct* password files; a user need never have an actual account on a given Unix system in order to be validated for access to files being served from that system and protected with HTTP–based authentication.

Protection by password; multiple users allowed.

> This document is accessible to user `rover` with password `bacon` and user `jumpy` with password `kibbles`.

Protection by network domain.

> This document is only accessible to clients running on machines inside domain `ncsa.uiuc.edu`.
>
> *Note for non–NCSA readers:* The `.htaccess` file used in this case is as follows:

```
AuthUserFile /dev/null
AuthGroupFile /dev/null
AuthName ExampleAllowFromNCSA
AuthType Basic

<Limit GET>
order deny,allow
deny from all
allow from .ncsa.uiuc.edu
</Limit>
```

Protection by network domain -- exclusion.

> This document is accessible to clients running on machines anywhere *but* inside

domain `ncsa.uiuc.edu`.

Note for NCSA readers: The `.htaccess` file used in this case is as follows:

```
AuthUserFile /dev/null
AuthGroupFile /dev/null
AuthName ExampleDenyFromNCSA
AuthType Basic

<Limit GET>
order allow,deny
allow from all
deny from .ncsa.uiuc.edu
</Limit>
```

General Information

There are two levels at which authentication can work: per–server and per–directory. This tutorial primarily covers per–directory authentication. Per–directory authentication means that users with write access to part of the filesystem that is being served can control access to their files as they wish. They need not have root access on the system or write access to the server's primary config files.

Access control for a given directory is controlled by a file named `.htaccess` that resides in that directory. The server reads this file on each access to a document in that directory (or documents in subdirectories).

By–Password Authentication: Step By Step

So let's suppose you want to restrict files in a directory called `turkey` to username `pumpkin` and password `pie`. Here's what to do:

Create a file called `.htaccess` in directory `turkey` that looks like this:

```
AuthUserFile /otherdir/.htpasswd
AuthGroupFile /dev/null
AuthName ByPassword
AuthType Basic

<Limit GET>
require user pumpkin
</Limit>
```

Note that the password file will be in another directory (`/otherdir`).

Also note that in this case there is no group file, so we specify `/dev/null` (the standard Unix way to say "this file doesn't exist").

`AuthName` can be anything you want. `AuthType` should always currently be `Basic`.

Create the password file `/otherdir/.htpasswd`.

The easiest way to do this is to use the `htpasswd` program distributed with NCSA httpd. Do this:

```
htpasswd –c /otherdir/.htpasswd pumpkin
```

Type the password –– `pie` –– twice as instructed.

Check the resulting file to get a warm feeling of self–satisfaction; it should look like this:

```
pumpkin:y1ia3tjWkhCK2
```

That's all. Now try to access a file in directory `turkey` –– Mosaic should demand a username and password, and not give you access to the file if you don't enter `pumpkin` and `pie`. If you are using a browser that doesn't handle authentication, you will not be able to access the document at all.

How Secure Is It?

The password is passed over the network *not encrypted* but *not as plain text* –– it is "uuencoded". Anyone watching packet traffic on the network will not see the password in the clear, but the password will be easily decoded by anyone who happens to catch the right network packet.

So basically this method of authentication is roughly as safe as `telnet`–style username and password security –– if you trust your machine to be on the Internet, open to attempts to `telnet` in by anyone who wants to try, then you have no reason not to trust this method also.

Multiple Usernames/Passwords

If you want to give access to a directory to more than one username/password pair, follow the same steps as for a single username/password with the following additions:

Add additional users to the directory's `.htpasswd` file.

Use the `htpasswd` command without the `–c` flag to additional users; e.g.:

```
htpasswd /otherdir/.htpasswd peanuts
htpasswd /otherdir/.htpasswd almonds
htpasswd /otherdir/.htpasswd walnuts
```

Create a group file.

Call it `/otherdir/.htgroup` and have it look something like this:

```
my–users: pumpkin peanuts almonds walnuts
```

`...` where `pumpkin`, `peanuts`, `almonds`, and `walnuts` are the usernames.

Then modify the `.htaccess` file in the directory to look like this:

```
AuthUserFile /otherdir/.htpasswd
AuthGroupFile /otherdir/.htgroup
AuthName ByPassword
AuthType Basic

<Limit GET>
require group my–users
```

```
</Limit>
```

Note that `AuthGroupFile` now points to your group file and that group `my-users` (rather than individual user `pumpkin`) is now required for access.

That's it. Now any user in group `my-users` can use his/her individual username and password to gain access to directory `turkey`.

CERN has extensive documents on http-based authentication. The URL is http://info.cern.ch/hypertext/WWW/AccessAuthorization/Overview.html.

- ## Graphical Information Map Tutorial

Introduction

This document is a step-by-step tutorial for designing and serving graphical maps of information resources. Through such a map, users can be provided with a graphical overview of any set of information resources; by clicking on different parts of the overview image, they can transparently access any of the information resources (possibly spread out all across the Internet).

First Steps

This tutorial assumes use of NCSA httpd (version 1.0a5 or later). Some other servers (e.g. Plexus) can also serve image maps, in server-specific ways; see the specific server's docs for more information.

Make sure you have a working NCSA httpd server installed and running.

Make sure you have write privileges to the server's `conf/imagemap.conf` config file.

Also make sure that the `imagemap` program is compiled and in the server's `htbin` directory.

This tutorial also assumes use of NCSA Mosaic for X version 2.0. Other clients that support inlined GIF images and HTTP/1.0 URL redirection will also work.

Your First Image Map

In this section we walk through the steps needed to get an initial image map up and running.

First, create an image.

There are a number of image creation and editing programs that will work nicely -- the one I use is called *xpaint* (you can find it on **ftp.x.org** in **/R5contrib**; **The important thing is that the image ends up in GIF format.**

A common scheme for an image map is a collection of rectangles and circles, each containing a short text description of some piece of information or some information server; interconnections are conveyed through lines or arcs. Try to keep the individual items in the map spaced out far enough so a user will clearly know what he or she is clicking on.

Second, create an image map file.

Here is what an image map file looks like:

```
default /X11/mosaic/public/none.html

rect http://cui_www.unige.ch/w3catalog    15,8    135,39
rect gopher://rs5.loc.gov/11/global       245,86  504,143
rect http://nearnet.gnn.com/GNN-ORA.html  117,122 175,158
```

The format is fairly straightforward. The first line specifies the default response (the file to be returned if the region of the image in which the user clicks doesn't correspond to anything).

Subsequent lines specify rectangles in the image that correspond to arbitrary URLs –– for the first of these lines, the rectangle specified by `15,8` (`x,y` of the upper–left corner, in pixels) and `135,39` (lower–right corner) corresponds to URL `http://cui_www.unige.ch/w3catalog`.

So, what you need to do is find the upper–left and lower–right corners of a rectangle for each information resource in your image map. A good tool to use for this is *xv* (also on **ftp.x.org** in **/contrib**)–– pop up the Info window and draw rectangles over the image with the middle mouse button.

It doesn't matter where you put your map file or what you name it. For the purposes of this example, let's assume it's called `/foo/sample.map`.

Third, tell your server about your image map file.

You do this by adding a file to the server's `conf/imagemap.conf` file. The line looks like this:

```
sample : /foo/sample.map
```

`...` where `sample` is the *symbolic name* for your image map and */foo/sample.map* is the actual name of your map file.

Fourth, create an HTML document that contains your map image.

An example follows:

```
Click on the information resource you wish to see: <P>

<A HREF="http://machine/htbin/imagemap/sample">
<img src="sample.gif" ismap>
</A> <P>
```

Note:

- `machine` is the name of the machine on which *your* HTTP server resides.
- `sample` is the *symbolic name* of your image map (from above).
- `sample.gif` is the name of your image (assuming, of course, that it's in the same directory on your server as the HTML file).

Fifth, try it out! Load the HTML file, look at the inlined image, click somewhere, and see what happens.

Subsequent Image Maps

You can serve as many image maps from a single server as you want. Just add lines to `conf/imagemap.conf` pointing to each image map file you create.

Real−World Examples

Following are examples of distributed image maps on servers in the real world; they may or may not work at any point in time. The URL for them is provided.

- Experimental Internet Resources Metamap
  http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Demo/metamap.html
- University of California Museum of Paleontology
  http://ucmp1.berkeley.edu/.
- National Institute of Standards and Technology
  http://www.nist.gov/welcome.html
- server map at NCHPC information server
  http://info.lcs.mit.edu/Info/structure.html

WAIS and HTTP Integration

Introduction

This document overviews existing methods for using WAIS as a back−end search engine for HTTP servers.

Information herein is currently experimental and may or may not work for you.

WAIS and Plexus

Plexus is a powerful Perl−based HTTP server written and maintained by Tony Sanders at BSDI. The URL's you might be interested in are:
http://www.bsdi.com/server/doc/plexus.html
http://www.cs.cmu.edu:8001/Web/People/rgs/perl.html

WAIS and GN

GN is a multi−protocol server written and maintained by John Franks at NWU. It is shipped with support for WAIS as a back−end search engine.

The URL's you might be interested in are:
http://hopf.math.nwu.edu/
http://hopf.math.nwu.edu:70/0h/docs/waisgn.guide

WAIS and NCSA httpd 1.0

Rob McCool has written a CGI script which allows NCSA httpd 1.0 as well as other CGI compliant servers to access a WAIS database in the same way that is mentioned in this document. The script is in the CGI archive. It contains instructions for setting it up under httpd 1.0.

The URL's you might be interested in are:
ftp://ftp.ncsa.uiuc.edu/Web/ncsa_httpd/cgi/wais.tar.Z

freeWAIS 0.202's URL Type

freeWAIS 0.202 is shipped with support for type "URL". Use of this type is a little tricky.

First, **Mosaic 2.0 doesn't know how to deal with this type directly**, but Mosaic 2.1 (when it is released) will.

Second, use of this type apparently implies overloading the "headline" of a WAIS hit with the URL. This is fine, except then the description that the user sees of a given document *is* the URL, and URLs are, as usual, pretty cryptic things to just throw in front of average users.

But anyway, here's how it works:

```
waisindex ... −t URL what−to−trim what−to−add ...
```

So what does that mean?

Well, first, `−t URL` tells `waisindex` to use type `URL` (note use of *lowercase* `−t` in this instance).

Second, `what−to−trim` and `what−to−add` are parameters that tell the indexer how to put together the URL that's returned as the result of a query.

Suppose your documents are normally stored in `/X11/mosaic/public`. Suppose also that these documents are normally served via a URL that begins with `http://wintermute.ncsa.uiuc.edu:8080`.

This means that a file stored as `/X11/mosaic/public/foo.html`, for example, is normally served as `http://wintermute.ncsa.uiuc.edu:8080/foo.html`.

The `waisindex` command you'd use in this case would be something like the following:

```
waisindex -d ~/localwais/sources/www -export
  -t URL /X11/mosaic/public http://wintermute.ncsa.uiuc.edu:8080
  /X11/mosaic/public/*.html
```

... where `~/localwais/sources/www` is the name of the WAIS index file and `/X11/mosaic/public/*.html` are the files you are indexing.

When queries are made on this database, the string `/X11/mosaic/public` is removed from the beginning of the filename of a matching file and the string `http://wintermute.ncsa.uiuc.edu:8080` is put in its place.

As per our previous example: `/X11/mosaic/public/foo.html` turns into `http://wintermute.ncsa.uiuc.edu:8080/foo.html` as the result of a WAIS hit.

As you can see, this is perfect -- the WAIS server passes back the exact same URL that would normally be used to access this file via HTTP. So, everything from relative hyperlinks to relative inlined image references in the file will work correctly when the file is retrieved.